CLAIMS

What is claimed is:

1. A method for enabling events in a COBOL program, the method comprising:

maintaining, in a COBOL program, a index including a process identifier and an event associated with a child process;

placing the child process in a wait state;

signaling, by the COBOL program, the child process to run using the process identifier and the event associated with the child process.

2. The method of Claim 1, wherein the COBOL program signals a technical layer using the process identifier and event associated with the child process and further wherein the technical layer signals the child process to run.

3. The method of Claim 1, wherein the index maintained by the COBOL program maintains a plurality of identifiers and a plurality of events associated with a plurality of child processes.

4. The method of Claim 1, wherein the child process is placed in the wait state by a technical layer.

5. The method of Claim 1, wherein the technical layer is further defined as a COBOL technical layer in communication with the COBOL program.

6. The method of Claim 1, wherein the technical layer is further defined as COBOL library including at least one routine callable by the COBOL program.

7. The method of Claim 1, wherein the technical layer is integral to the COBOL program.

8. The method of Claim 1, wherein the technical layer is enabled by a COBOL compiler.

9. The method of Claim 1, wherein the technical layer is integral to a COBOL compiler.

10. The method of Claim 1, wherein the technical layer includes a coordination module operable.

11. The method of Claim 1, wherein the child process registers the process identifier of the child process with a technical layer.

12. The method of Claim 11, wherein the child process further registers the event associated with the child process with the technical layer.

13. The method of claim 1, further comprising maintaining a plurality of child processes wherein the process identifiers and events associated with each of the plurality of child processes is maintained in the index of the COBOL program.

14. The method of 13, further comprising:

providing a COBOL technical layer having a coordination module operable to coordinate signaling the plurality of child processes;

registering, by the plurality of child processes, with the COBOL technical layer;

signaling, by the COBOL program, the COBOL technical layer to run one or more of the plurality of child processes using the process identifiers and events associated with the child processes; and

coordinating, by the coordination module of the COBOL technical layer, the signaling of the child processes.

15. The method of Claim 14, wherein method further comprises:

creating a system resource by the COBOL program;

designating the system resource to a process identification of the COBOL program;

giving the system resource from the COBOL program to the child process using the process identifier of the child process; and

taking the system resource by the child process from the COBOL program.

16. The method of Claim 15, further comprising synchronizing such that the COBOL program completes giving the system resource prior to the child process taking the system resource.

17. The method of Claim 16, wherein the system resource is defined as a socket connection.

18. The method of Claim 16, wherein the system resource is defined as a pipe connection.

19. The method of Claim 16, further comprising:

placing the COBOL program in a wait state after giving the system resource to the child process; and

maintaining the COBOL program in the wait state until the child process takes the system resource.

20. A system for coordinating processing in COBOL programs, comprising:

a first COBOL program having a first routine for processing;

a second COBOL program having a second routine for processing; and

a module callable by the first and second COBOL programs, the module maintaining a state sharable between the first and second COBOL programs to coordinate the processing of the first and second routines.

21. The system of Claim 20, wherein the module is further defined as a COBOL library having routines callable from the first and second COBOL programs.

22. The system of Claim 20, wherein the module is further defined as a COBOL callable routine.

23. The system of Claim 20, wherein the module is integral to the first and second COBOL programs.

24. The system of Claim 20, wherein the module is further defined as COBOL compiler enabled.

25. The system of Claim 20, wherein the module is further defined as a COBOL pre-compiler program used by the first and second COBOL programs.

26. The system of Claim 20, wherein the first and second routines are further defined as a first and second jobs and wherein the first and second COBOL programs analyze the state maintained by the technical layer to resolve the processing of the first and second jobs.

27. The system of Claim 20, wherein the module initiates a first state when the first routine is processing such that the second routine postpones processing in response to the first state of the module.

28. The system of Claim 20, wherein the module is operable to maintain a plurality of states based upon a task of the first and second routines.

29. The system of Claim 20, wherein the first and second routines are further defined as a first and second threads and wherein the module maintains the state sharable between the first and second COBOL programs to coordinate the processing of the first and second threads.

30. The system of Claim 29, wherein the first and second threads process in the same address space in a computer system.

31. The system of Claim 20, wherein the first and second routines are further defined as a first and second jobs and wherein the module maintains the state sharable between the first and second COBOL programs to coordinate the processing of the first and second jobs.

32. The system of Claim 31, wherein the first and second jobs process in separate address space in a computer system.

33. A method for employing semaphores to coordinating processing in COBOL programs, comprising:

processing by a first COBOL program to a shared resource;

processing by a second COBOL program to the shared resource; and

maintaining a state sharable between the first and COBOL programs to coordinate the processing by of the first and second COBOL programs to the shared resource.

34. The method of Claim 33, further comprising providing a COBOL technical layer operable to maintain the sharable state.

35. The method of Claim 34, further comprising:

creating a semaphore, by the COBOL technical layer;

obtaining an identifier identifying the semaphore;

querying to determine whether the state indicates that the semaphore is locked;

changing the state to indicate that the semaphore is locked;

changing the state to indicate that the semaphore is unlocked;

obtaining a process identification number to determine a process associated with the semaphore; and

removing the semaphore from a computer system.

36. The method of Claim 35, wherein the state further defined as is a flag.

37. A method of employing threads in COBOL programs, comprising:

outputting by a first COBOL program to a block of shared memory;

outputting by a second COBOL program to the block of share memory;

writing, by a COBOL routine, the output of the first COBOL program to a shared resource; and

writing, by the COBOL routine, the output of the second COBOL program to the shared resource.

38. The method of Claim 37, wherein the method includes creating the block of shared memory.

39. The method of Claim 37, further comprising:

passing the output of the first COBOL program to the COBOL routine;

writing, by the COBOL routine, the output of the first COBOL program to the block of shared memory;

passing the output of the second COBOL program to the COBOL subroutine; and

writing, by the COBOL subroutine, the output of the second COBOL program to the block of shared memory.

40. The method of Claim 37, wherein the first and second COBOL programs are further defined as a first and second COBOL subtasks.

41. The method of Claim 37, wherein the COBOL routine includes a coordination module and wherein the method further comprises:

gathering output from the first and second COBOL programs; and

coordinating the writing of the output from the first and second COBOL programs to the shared resource.

42. The method of Claim 41, wherein the technical layer is further defined as a COBOL library callable from the first and second COBOL programs.

43. The method of Claim 41, wherein the COBOL routine is integral to the first and second COBOL programs.

44. The method of Claim 41, wherein the COBOL routine is further defined as a compiler enabled function.

45. The method of Claim 37, wherein the share resource is further defined as a socket connection.

46. The method of Claim 37, wherein the shared resource is further defined as a display resource.

47. A method for a COBOL program to use signal handlers, the method comprising:

registering, by a COBOL language program, a signal handler with an operating system, the signal handler associated with an event; and

executing, by the operating system, the signal handler on the event occurs.

48. The method of Claim 47, wherein the COBOL language program registers with a register of the operating system.

49. The method of Claim 47, wherein the signal handler executes a corrective process.

50. The method of Claim 49, wherein the corrective process is closing a file.

51. The method of Claim 47, wherein the signal handler executes a notification process.

52. The method of Claim 47, wherein the event is an input/output error.

53. The method of Claim 47, further comprising:

creating a memory block:

writing an identifier to the memory block related to a system processes being executed; and

reading from the memory block the identifier to determine the system process executed when the event occurred.

54. A system for coordinating processing in COBOL, comprising:

a COBOL program desiring to process a first and second tasks to a shared resource;

a module in communication with the COBOL program and maintaining a shared state between the first and second tasks to coordinate processing to the shared resource, the COBOL program and module operating in the same runtime environment.